Prem Mody **IT A**

ERROR BACKPROPAGATION TRAINING ALGORITHM

Source Code:

```
import numpy as np

train_data = np.loadtxt("data_train.csv",
                        delimiter=",")
test_data = np.loadtxt("data_test.csv",
                       delimiter=",")


#print(train_data[0])
image_size = 28 # width and length
no_of_different_labels = 10
image_pixels = image_size * image_size


train_labels = np.asfarray(train_data[:, :1])
test_labels = np.asfarray(test_data[:, :1])
lr = np.arange(no_of_different_labels)


# transform labels into one hot representation

train_labels_one_hot = (lr==train_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)


def sigmoid(x):
    return 1 / (1 + np.e ** -x)
activation_function = sigmoid


from scipy.stats import truncnorm
```

```
def truncated_normal(mean=0, sd=1, low=0, upp=10):
   return truncnorm((low - mean) / sd,
                    (upp - mean) / sd,
                    loc=mean,
                    scale=sd)
class NeuralNetwork:

   def __init__(self,
                no_of_in_nodes,
                no_of_out_nodes,
                no_of_hidden_nodes,
                learning_rate):
      self.no_of_in_nodes = no_of_in_nodes
      self.no_of_out_nodes = no_of_out_nodes
      self.no_of_hidden_nodes = no_of_hidden_nodes
      self.learning_rate = learning_rate
      self.create_weight_matrices()

   def create_weight_matrices(self):
      rad = 1 / np.sqrt(self.no_of_in_nodes)
      X = truncated_normal(mean=0,
                           sd=1,
                           low=-rad,
                           upp=rad)
      self.wih = X.rvs((self.no_of_hidden_nodes,
                              self.no_of_in_nodes))
      rad = 1 / np.sqrt(self.no_of_hidden_nodes)
      X = truncated_normal(mean=0, sd=1, low=-rad, upp=rad)
      self.who = X.rvs((self.no_of_out_nodes,
                               self.no_of_hidden_nodes))


   def train(self, input_vector, target_vector):
      input_vector = np.array(input_vector, ndmin=2).T
      target_vector = np.array(target_vector, ndmin=2).T

      output_vector1 = np.dot(self.wih,
                              input_vector)
      output_hidden = activation_function(output_vector1)

      output_vector2 = np.dot(self.who,
                              output_hidden)
      output_network = activation_function(output_vector2)
```

```
    output_errors = target_vector - output_network
    # update the weights:
    tmp = output_errors * output_network \
        * (1.0 - output_network)
    tmp = self.learning_rate  * np.dot(tmp,
                                output_hidden.T)
    self.who += tmp
    # calculate hidden errors:
    hidden_errors = np.dot(self.who.T,
                           output_errors)
    # update the weights:
    tmp = hidden_errors * output_hidden * \
        (1.0 - output_hidden)
    self.wih += self.learning_rate \
                * np.dot(tmp, input_vector.T)



def run(self, input_vector):
    # input_vector can be tuple, list or ndarray
    input_vector = np.array(input_vector, ndmin=2).T
    output_vector = np.dot(self.wih,
                           input_vector)
    output_vector = activation_function(output_vector)

    output_vector = np.dot(self.who,
                           output_vector)
    output_vector = activation_function(output_vector)

    return output_vector

def evaluate(self, data, labels):
    corrects, wrongs = 0, 0
    for i in range(len(data)):
        res = self.run(data[i])
        res_max = res.argmax()
        if res_max == labels[i]:
            corrects += 1
        else:
            wrongs += 1
    return corrects, wrongs
```

```
ANN = NeuralNetwork(no_of_in_nodes = image_pixels,
                    no_of_out_nodes = 10,
                    no_of_hidden_nodes = 100,
                    learning_rate = 0.01)


for i in range(len(train_data)):
    ANN.train(train_data[i], train_labels_one_hot[i])

for i in range(10):
    res = ANN.run(test_data[i])

corrects, wrongs = ANN.evaluate(train_data, train_labels)
print("Accuracy train:{}% ".format( corrects / ( corrects + wrongs)*100))
corrects, wrongs = ANN.evaluate(test_data, test_labels)
print("Accuracy: test:{}%".format(corrects / ( corrects + wrongs)*100))
```

Output:

```
In [1]: runfile('E:/EBPTA/EBPTA.py', wdir='E:/EBPTA')
Accuracy train:91.0%
Accuracy: test:90.0%
```